tsi Documentation

Release 0.1

Phil Maker

January 18, 2016

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

CHAPTER

TWO

SETUP, CONVENTIONS AND TOOLS

2.1 Conventions

1. Use Python version 3 1. Use the pandas, numpy,scipy and matplotlib 1. Documentation and tests will in Sphinx using the numpydoc

extension.

2.2 Setup

1. Install Sphinx
2. Configure the documentation in the doc directory using `spinx-quickstart` to configure it. Select the following options:
   (a) autodoc
   (b) doctest
   (c) document coverage
   (d) pngmath
   (e) viewcode
3. Install `numpydoc` and add into the extensions list in ‘‘conf.py’‘
4. Add a reference to the various `intro.rst` and `history.rst` into `index.rst`
5. Add `numpydoc` to the modules, e.g. `tsi.py`.

CHAPTER

THREE

MODULES

3.1 tsi

This is a useful module which I am currently describing in the index.rst file. Example Google style docstrings.

This module demonstrates documentation as specified by the Google Python Style Guide. Docstrings may extend over multiple lines. Sections are created with a section header and a colon followed by a block of indented text.

Example

Examples can be given using either the `Example` or `Examples` sections. Sections support any reStructuredText formatting, including literal blocks:

```
$ python example_google.py
```

Section breaks are created by resuming unindented text. Section breaks are also implicitly created anytime a new section starts.

`tsi.`**`module_level_variable1`**
int

Module level variables may be documented in either the `Attributes` section of the module docstring, or in an inline docstring immediately following the variable.

Either form is acceptable, but the two should not be mixed. Choose one convention to document module level variables and be consistent with it.

class `tsi.`**`ExampleClass`**(*param1*, *param2*, *param3*)
The summary line for a class docstring should fit on one line.

If the class has public attributes, they may be documented here in an `Attributes` section and follow the same formatting as a function's `Args` section. Alternatively, attributes may be documented inline with the attribute's declaration (see __init__ method below).

Properties created with the `@property` decorator should be documented in the property's getter method.

Attribute and property types – if given – should be specified according to PEP 484, though PEP 484 conformance isn't required or enforced.

`attr1`
str

Description of *attr1*.

`attr2`
Optional[int]

Description of *attr2*.

Attributes

Methods

__special__()
By default special members with docstrings are included.

Special members are any methods or attributes that start with and end with a double underscore. Any special member with a docstring will be included in the output.

This behavior can be disabled by changing the following setting in Sphinx's conf.py:

```
napoleon_include_special_with_doc = False
```

attr3 = None
Doc comment *inline* with attribute

attr4 = None
List[str]: Doc comment *before* attribute, with type specified

attr5 = None
Optional[str]: Docstring *after* attribute, with type specified.

example_method(*param1*, *param2*)
Class methods are similar to regular functions.

Note: Do not include the *self* parameter in the `Args` section.

Parameters

- **param1** – The first parameter.
- **param2** – The second parameter.

Returns True if successful, False otherwise.

readonly_property
str: Properties should be documented in their getter method.

readwrite_property
List[str]: Properties with both a getter and setter should only be documented in their getter method.

If the setter method contains notable behavior, it should be mentioned here.

exception `tsi.`**ExampleError**(*msg*, *code*)
Exceptions are documented in the same way as classes.

The __init__ method may be documented in either the class level docstring, or as a docstring on the __init__ method itself.

Either form is acceptable, but the two should not be mixed. Choose one convention to document the __init__ method and be consistent with it.

Note: Do not include the *self* parameter in the `Args` section.

Parameters

- **msg** (*str*) – Human readable string describing the exception.
- **code** (*Optional[int]*) – Error code.

msg
str

Human readable string describing the exception.

code
int

Exception error code.

tsi.**example_generator**(*n*)
Generators have a `Yields` section instead of a `Returns` section.

Parameters **n** (*int*) – The upper limit of the range to generate, from 0 to *n* - 1.

Yields *int* – The next number in the range of 0 to *n* - 1.

Examples

Examples should be written in doctest format, and should illustrate how to use the function.

```
>>> print([i for i in example_generator(4)])
[0, 1, 2, 3]
```

tsi.**main**()
The main program

tsi.**module_level_function**(*param1*, *param2=None*, **args*, ***kwargs*)
This is an example of a module level function.

Function parameters should be documented in the `Args` section. The name of each parameter is required. The type and description of each parameter is optional, but should be included if not obvious.

Parameter types – if given – should be specified according to PEP 484, though PEP 484 conformance isn't required or enforced.

If *args or **kwargs are accepted, they should be listed as `*args` and `**kwargs`.

The format for a parameter is:

```
name (type): description
    The description may span multiple lines. Following
    lines should be indented. The "(type)" is optional.

    Multiple paragraphs are supported in parameter
    descriptions.
```

Parameters

- **param1** (*int*) – The first parameter.
- **param2** (*Optional[str]*) – The second parameter. Defaults to None. Second line of description should be indented.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns

True if successful, False otherwise.

The return type is optional and may be specified at the beginning of the `Returns` section followed by a colon.

The `Returns` section may span multiple lines and paragraphs. Following lines should be indented to match the first line.

The `Returns` section supports any reStructuredText formatting, including literal blocks:

```
{
    'param1': param1,
    'param2': param2
}
```

Return type bool

Raises

- `AttributeError` – The `Raises` section is a list of all exceptions that are relevant to the interface.
- `ValueError` – If *param2* is equal to *param1*.

`tsi.`**module_level_variable2 = 98765**

int: Module level variable documented inline.

The docstring may span multiple lines. The type may optionally be specified on the first line, separated by a colon.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

Symbols

A

C

E

M

R

T